Calcolo del costo medio di BUBBLE SORT. Al fine di quantificare il costo $T_{average}(n)$ nel caso medio del Bubble Sort, iniziamo quantificando un limite inferiore al numero medio di swap in cui viene coinvolto ogni valore in input, durante l'esecuzione dell'algoritmo.

Consideriamo un generico valore inizialmente in posizione i nell'array in input. Sia j la posizione di quel valore nell'array finale ordinato. Il numero di swap in cui viene coinvolto quel valore sarà inferiormente limitato dalla distanza fra i e j. Il valore di j può essere un qualsiasi valore nell'intervallo $[1, n]$, insieme dei possibili indici dell'array, ed è ragionevole assumere che tutti questi indici siano ugualmente frequenti.

A titolo di esempio, consideriamo $i = n/2$, ovvero consideriamo che i sia l'indice della cella centrale dell'array. La distanza massima del valore inizialmente in posizione $n/2$ rispetto alla sua posizione nell'array finale ordinato è $n/2$, mentre la distanza media sarà $\frac{n}{4}$, sotto l'assunzione di uguale frequenza descritta sopra. Quindi $\frac{n}{4}$ è un limite inferiore al numero medio di swap in cui il valore inizialmente in posizione $i = n/2$ viene coinvolto. Gli altri valori non in posizione centrale subiranno un numero medio di swap superiore, in quanto potrebbero inizialmente trovarsi ad una distanza superiore a $n/2$ dalla loro collocazione finale. Quindi $\frac{n}{4}$ è un limite inferiore al numero medio di swap in cui ogni valore viene coinvolto durante l'esecuzione di Bubble Sort.

Ora osserviamo che ogni volta che un valore viene coinvolto in uno swap, genera almeno una operazione di assegnamento per inserire tale valore in una nuova cella dell'array. Quindi, moltiplicando per n (numero di valori) il limite inferiore $\frac{n}{4}$ al numero medio di swap per ogni valore, otteniamo un limite inferiore al numero medio di operazioni elementari di assegnamento eseguite da Bubble Sort, e quindi anche un limite inferiore al costo computazionale $T_{average}(n)$. Abbiamo quindi che $n \times \frac{n}{4} \leq T_{average}(n)$.

Sapendo che $n \times \frac{n}{4} = \Theta(n^2)$ possiamo concludere che $\Theta(n^2) \leq T_{average}(n)$. Visto che nel caso pessimo Bubble Sort ha costo quadratico, possiamo osservare che $T_{average}(n) \leq T_{worst}(n) = \Theta(n^2)$. Mettendo insieme $\Theta(n^2) \leq T_{average}(n)$ e $T_{average}(n) \leq \Theta(n^2)$, concludiamo che $T_{average}(n) = \Theta(n^2)$.

Il problema della bandiera nazionale. Supponiamo di avere un array $A[1..n]$ di elementi che possono assumere solo tre valori: *bianco*, *verde* e *rosso*. Ordinare l'array in modo che tutti gli elementi *verdi* siano a sinistra, quelli *bianchi* al centro e quelli *rossi* a destra. L'algoritmo DEVE richiedere tempo $O(n)$ e memoria aggiuntiva $O(1)$. Può confrontare ed eventualmente scambiare tra loro elementi, e NON DEVE fare uso di ulteriori array di appoggio, né usare contatori per tenere traccia del numero di elementi di un certo colore. L'algoritmo DEVE richiedere una singola scansione dell'array.

Soluzione Si può usare un algoritmo che utilizza tre indici i, j e k in modo tale che la seguente proprietà valga per tutta l'esecuzione dell'algoritmo (per questo motivo, questa proprietà viene detta *invariante*):

> *tutti gli elementi in posizioni minori di i sono verdi, tutti gli elementi in posizioni minori di j non sono rossi (quindi sono verdi o bianchi), e tutti gli elementi in posizioni maggiori di k sono rossi*

Inizialmente poniamo $i = 1$, $j = 1$ e $k = n$ e notiamo che l'invariante vale banalmente in quanto non ci sono elementi in posizioni minori di i e j o in posizioni maggiori di k. Successivamente si procede in modo iterativo come segue:

- se in posizione j c'è il colore *verde*, scambio i colori in posizione i e j (se $i \neq j$) ed incremento entrambi gli indici i e j;
- se in posizione j c'è il colore *rosso*, scambio i colori in posizione j e k e decremento l'indice k;
- se invece in posizione j c'è il colore *bianco* semplicemente incremento j.

Si noti che procedendo in questo modo l'invariante continua a valere. L'algoritmo termina quando j e k si incontrano, ovvero $j == k$; questa condizione prima o poi si verifica in quanto ad ogni iterazione i viene incrementato oppure (*or* esclusivo) k viene decrementato. Quando $j == k$, l'invariante garantisce che nelle posizioni $1..i-1$ avremo solo il *verde*, nelle posizioni $i..j-1$ avremo solo il *bianco*, e nelle posizioni $j..n$ avremo solo il *rosso*.

L'algoritmo scritto in pseudocodice è riportato in tabella Algorithm 1.

Tale algoritmo richiede tempo lineare $\Theta(n)$ in quanto tutte le operazioni hanno costo costante ed il ciclo while viene eseguito esattamente $n - 1$ volte visto che inizialmente $k - j = n - 1$, ad ogni ciclo la differenza $k - j$ si decrementa di 1, e si esce dal ciclo quando questa differenza diventa 0. L'algoritmo utilizza una quantità costante $O(1)$ di memoria aggiuntiva: le tre variabili i, j, k ed una variabile di appoggio per gli scambi.

Algorithm 1: BandieraNazionale($A[1..n]$)

Input: Una sequenza $A[1..n]$ contenente valori che possono essere *bianco*, *verde*, *rosso*
Output: La sequenza $A[1..n]$ con i valori *verde* all'inizio, poi i valori *bianco*, ed infine i valori *rosso*

```
i ← 1; j ← 1; k ← n
while j < k do
    if A[j] = verde then
        if i < j then
            scambia A[i] con A[j]
        i ← i + 1; j ← j + 1
    else if A[j] = rosso then
        scambia A[j] con A[k]
        k ← k − 1
    else
        /* quindi A[j] = bianco                                   */
        j ← j + 1
return A
```

The End.